COM SCI 118 Computer Network Fundamentals

Concurrent Web Server using BSD Sockets

Fall 2015

Aaron Cheng
UID: 003 933 507
SEASNET login name: aaronc

Guan Zhou
UID: 604 501 750
SEASNET login name: guan

Goal

The goal of this project was to develop a web server in C using BSD sockets, using this opportunity to learn how a server and client communicate with each other and how data is transferred between them.

High-Level Description

The web server we created used the server demo, from the cs118 image home folder (/home/cs118/workspace/ClientServer_Example/serverFork.c), as a skeleton code. Like the demo, instead of specifying a static port number in the code, our executable takes as input a port number to listen to.

The browser we used was Mozilla Firefox. After running the server with a selected port number, a client, such as Firefox, can connect to the server via the corresponding port number. With the listen() function, the server will accept all successful connections and start TCP data transmission. For fork(), the pid of the child process is returned in the parent's thread of execution on success, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, and no child process will be created. As shown in the main function, by creating new process and checking the pid, we can detect when the connections need to be handled.

```
pid = fork(); //create a new process
if (pid < 0)
       error("ERROR on fork");

if (pid == 0)  { // fork() returns a value of 0 to the child process
       close(sockfd);
       handle_connection(newsockfd);
       exit(0);
}
else //returns the process ID of the child process to the parent
       close(newsockfd); // parent doesn't need this
```

The handle_connection() function reads in the HTTP request message from the socket, prints a copy of the message to the console, parses through the message for the requested file, and in turn calls the send_request() function. The send_request() function constructs the appropriate HTTP response message (dependent on the file requested), sends the message along with any files, and prints a copy of the response message to the console. One of the parameters in the send_header() function is the status code, which are 200 and 404, which we assign based on the cases that may occur, such as empty file name, not finding files, etc. The constructed HTTP response message format was referred from textbook page 105-106, and we used six buffers to hold different sections of the header: the connection, current date/time, server, last_modified time, content length, and content type.

Difficulties We Faced and the Ways We Solved Them

For determining the type of the file that was requested, initially our idea was to check if the filename contained any of the supported file extensions, like jpeg, html, and gif. Therefore, we started with the strstr() function. Later, we realized the drawback of this function when requested files contained two or more extensions, such as “file.html.jpeg.” We ended up using the function strrchr(const char *s, int c), which locates the last occurrence of c (converted to a char) in the string pointed to by s.

Another difficulty we met was that we failed several times when serving the html files. For the same html file, we switched to other port numbers and sometimes it worked and sometimes it failed. Later we found that this was related to the browser cache. What we did next was to clear the cache, and we saw all the files were successfully served to the clients.

How to Compile and Run the Source Code

Instead of specifying the port number in the code, we made choosing port number more flexible. The first step was to generate the executable file. In the linux terminal, this can be done using the below command:

```
  gcc webserver.c -o webserver
```

This will generate the executable file. Suppose that the executable is in the current directory, then we can run the server with command: ./webserver <port number>
For example, if we choose our port number to be 1025, then we can start the server as follows:

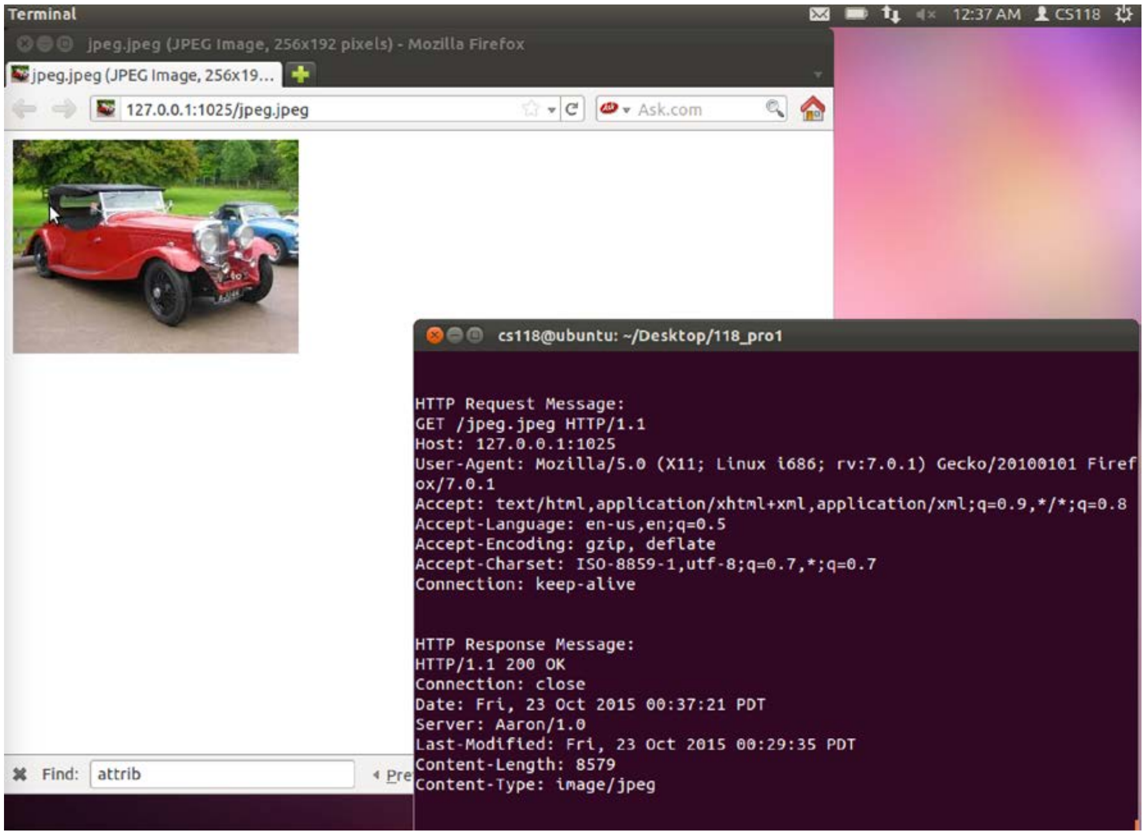
```
  ./webserver 1025
```

The server will be started and will listen to the connections from clients such as Firefox.

Sample Outputs of the Client-Server

After the server is running, we can test it in Firefox browser. We prepared three files, file.html, jpeg.jpeg, and gif.GIF. One thing that needs to be careful is that all these requested files must be in the same directory as the webserver. Also, similar as to what the textbook provided, we set text/html as the default type. The port number we chose is 1025.

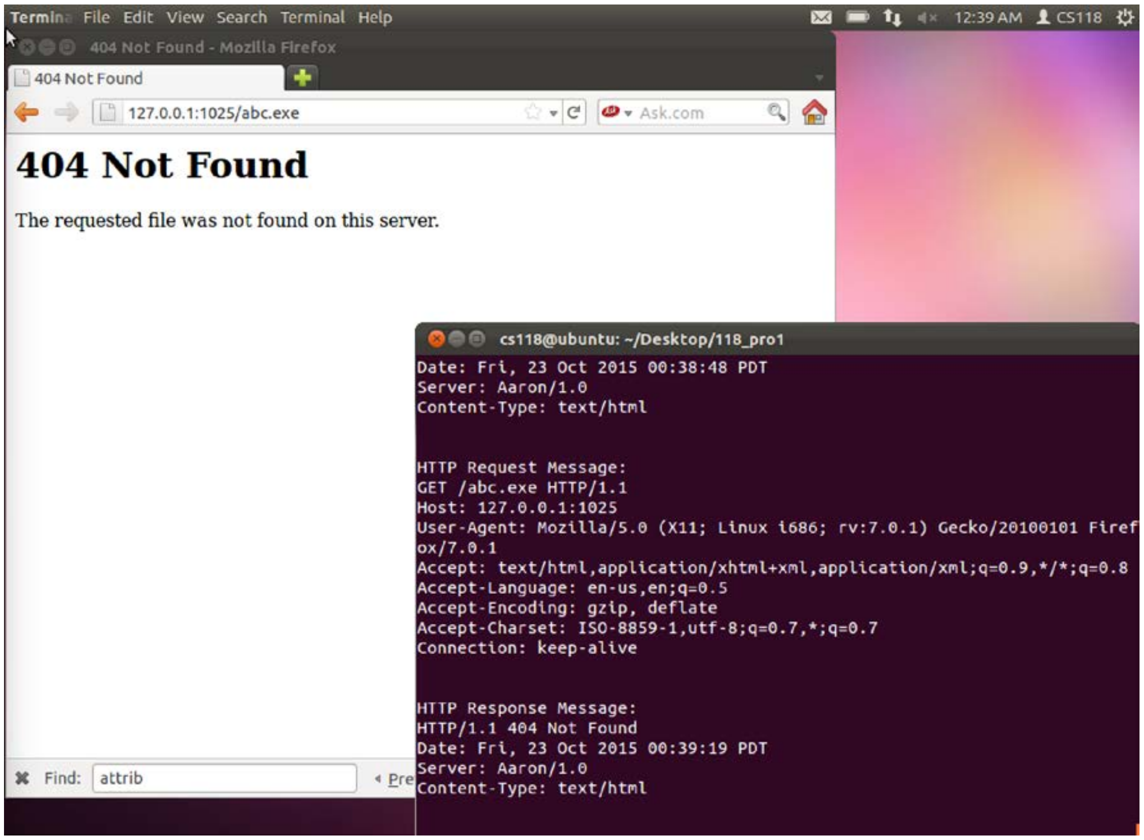
Below is an example when file jpeg.jpeg was served:


Terminal
12:37 AM
CS118
jpeg.jpeg (JPEG Image, 256x192 pixels) - Mozilla Firefox
jpeg.jpeg (JPEG Image, 256x19...
127.0.0.1:1025/jpeg.jpeg
Ask.com
Find: attrib
cs118@ubuntu: ~/Desktop/118_pro1

```
HTTP Request Message:
GET /jpeg.jpeg HTTP/1.1
Host: 127.0.0.1:1025
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:7.0.1) Gecko/20100101 Firef
ox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive


HTTP Response Message:
HTTP/1.1 200 OK
Connection: close
Date: Fri, 23 Oct 2015 00:37:21 PDT
Server: Aaron/1.0
Last-Modified: Fri, 23 Oct 2015 00:29:35 PDT
Content-Length: 8579
Content-Type: image/jpeg
```

If file name is not provided, specified file is not found, or file type is not supported, then 404 status error will be returned.


Terminal File Edit View Search Terminal Help
12:39 AM CS118
404 Not Found - Mozilla Firefox
404 Not Found
127.0.0.1:1025/abc.exe
Ask.com
Find: attrib
cs118@ubuntu: ~/Desktop/118_pro1

404 Not Found

The requested file was not found on this server.

```
Date: Fri, 23 Oct 2015 00:38:48 PDT
Server: Aaron/1.0
Content-Type: text/html


HTTP Request Message:
GET /abc.exe HTTP/1.1
Host: 127.0.0.1:1025
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:7.0.1) Gecko/20100101 Firef
ox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive


HTTP Response Message:
HTTP/1.1 404 Not Found
Date: Fri, 23 Oct 2015 00:39:19 PDT
Server: Aaron/1.0
Content-Type: text/html
```

Conclusion

First of all, we obtained a deeper understanding how server and client communicate with each other as well as how data is transferred between them through this project. Also, we learned some powerful C libraries and function that we never met before.